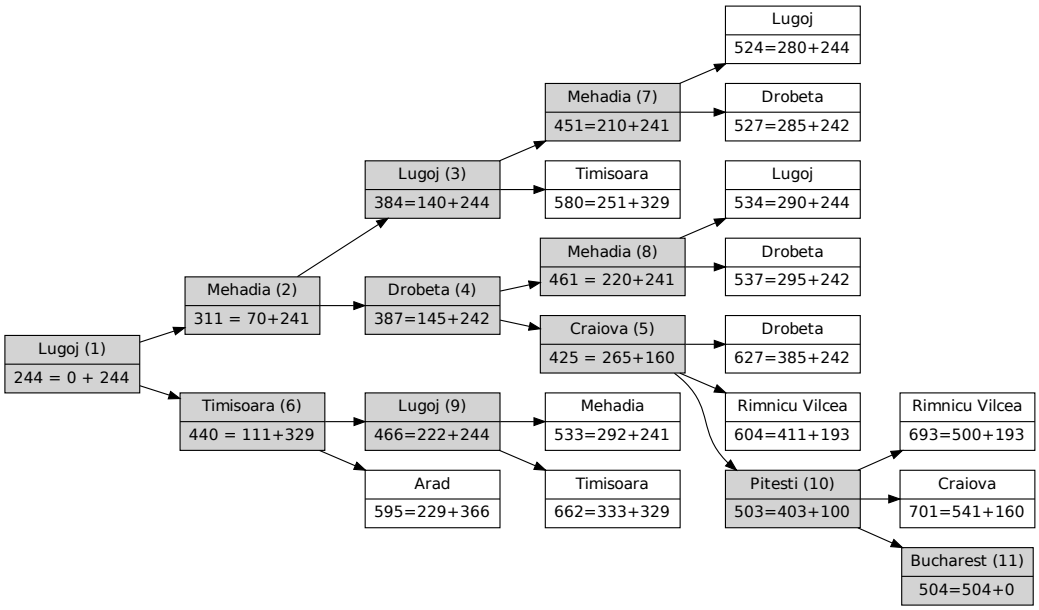
Einführung in die
Künstliche Intelligenz
SS 17- Prof. Dr. J. Fürnkranz

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiellösung für das 2. Übungsblatt

Aufgabe 1 Informierte Suchalgorithmen

a) In dem folgenden Graph sind die expandierten Knoten grau markiert und die Reihenfolge, in der sie expandiert werden, ist durch die Nummerierung dargestellt.



b) Man kann bei dieser Aufgabe den Algorithmus graphisch durchführen, da der Graph in etwa die tatsächliche geometrische Lage der Städte darstellt, so dass die angegebenen Distanzen und die Heuristiktabelle mit dem Graph bis auf einen Skalierungsfaktor konsistent sind.

Greedy Best First-Search verfährt hier nach folgender Regel: Expandiere den Knoten aus der Kandidatenmenge mit der minimalen Entfernung (im Graph) zu Bucharest. So lassen sich recht schnell nahezu alle Städte überprüfen, ohne einen Blick auf die Heuristiktabelle zu werfen. Ein Fall, bei dem es aus dem Graph eventuell nicht ganz klar ist, tritt bei der Expansion von Mehadia auf, da Lugoj und Drobeta etwa gleich weit entfernt von Bucharest erscheinen.

Für alle Städte findet Greedy Best-First Search eine Verbindung nach Bucharest.

c) Auch hier lässt sich das Problem schnell graphisch lösen. Für die Städte rechts von Bucharest und Giurgiu sind die gefundenen Verbindungen klar die kürzesten, da keine anderen Verbindungen (ohne mehrfache Benutzung von Kanten) existieren.

Die Verbindungen von den restlichen Städte lassen sich mithilfe kleiner Überlegungen ähnlich schnell auf Optimalität prüfen. Hilfreich ist z.B. die notwendige Bedingung, dass jede Teilverbindung einer optimalen Verbindung wieder eine optimale Verbindung sein muss. Zum Beispiel ist die Verbindung von Sibiu nach Bucharest, die Greedy Best-First Search findet (Sibiu, Fagaras, Bucharest), nicht optimal. Somit sind auch die ermittelten Verbindungen von Oradea, Zerind und Arad, die durch Sibiu verlaufen, auch nicht optimal.

Für Timisoara, Arad, Zerind, Oradea und Sibiu findet Greedy Best-First Search nicht die optimale Verbindung.

a) Eine Heuristik ist konsistent, falls $h(n) \leq c(n,a,n') + h(n')$ gilt.

Zunächst ist festzustellen, dass die Kosten für eine Aktion (Zug) hier auf 1 gesetzt sind, d.h. für alle legalen Züge ist $c(n,a,n') = 1$. Für beide Heuristiken muss also gezeigt werden: $h(n) \leq 1 + h(n')$.

Die h_{MIS} Heuristik zählt die Anzahl falsch platzierter Steine. Mit einem Zug kann sich die Anzahl der falsch platzierten Steine um maximal eins verringern, d.h. $h(n') \geq h(n) - 1$.

Die h_{MAN} Heuristik zählt die Anzahl der Felder, die überquert werden müssen, um zur Zielposition zu gelangen (summiert über alle Steine). Mit einem Zug ändert sich die entsprechende Anzahl nur für ein Stein, die nur maximal um eins verringert sein kann.

b) Seien h, h' zwei beliebige admissible Heuristiken bezüglich der wahren Kostenfunktion h^*. Es gilt also: $h(n) \leq h^*(n)$ und $h'(n) \leq h^*(n)$ für alle n.

Sei g nun die mittels dem Maximum-Operator kombinierte Heuristik von h und h', d.h. $g(n) := max(h(n), h'(n))$. Es gilt offensichtlich $g(n) \leq h^*(n)$, da für beliebige n, die Heuristik g entweder $h(n)$ oder $h'(n)$ zurück liefert, welche beide per Konstruktion nicht größer als $h^*(n)$ sind.

a) Hill Climbing

b) Simulated Annealing nimmt eine Lösung an, wenn sie besser ist ($\Delta E > 0$), oder mit einer Wahrscheinlichkeit $e^{\Delta E/T}$, wenn sie schlechter ist.

   Für $\Delta E < 0$ wird die Wahrscheinlichkeit $e^{\Delta E/T}$ sich an 0 annähern, d.h. suboptimale Aktionen werden praktiksch nie durchgeführt werden → (ähnlich zu) Hill Climbing

c) Für $\Delta E < 0$ wird die Wahrscheinlichkeit $e^{\Delta E/T}$ sich an 1 annähern, d.h. suboptimale Aktionen werden praktiksch immer durchgeführt werden → (ähnlich zu) Random Walk, eine zufällige Traversierung des Zustandsraumes.

Es sollte klar sein, dass folgende Formalisierung eine Möglichkeit und nicht die einzig Richtige darstellt.

a) Wertebereiche der Variablen:

$$\begin{aligned} domain(A) = domain(C) &= \{0,\ldots,9\} \\ domain(B) &= \{1,\ldots,9\} \\ domain(U) &= \{0,1\} \end{aligned}$$

Constraints:

$$\begin{aligned} A + B &= C + 10 \cdot U \\ B &= U \\ A \neq B,\ A \neq C,\ B &\neq C \end{aligned}$$

b)
```
C selected
  C = 0 consistent
    B selected
      B = 1 consistent
        A selected
          A = 0 not consistent (A != C constraint violated)
          A = 1 not consistent (A != B constraint violated)
          A = 2 consistent
            U selected
              U = 0 not consistent (B = U constraint violated)
              U = 1 not consistent (A+B=C+10*U constraint violated)
            U removed
          A = 3 consistent
          ... (identischer Verlauf wie in A=2, mit den gleichen Const
          A = 9 consistent
            U selected
              U = 0 not consistent
              U = 1 consistent
              result <- C=0,B=1,A=9,U=1
```

c) Forward Checking erweitert Backtracking dahingehend, daß wenn immer eine neue Variable X einen Wert erhält, es überprüft wird, ob es für jeden Wert einer durch einen Constraint mit X verbundene Variable Y noch einen passenden Wert von X gibt. Falls nicht, wird der entsprechende Wert aus Y gelöscht.

In der folgenden Darstellung des Algorithmus-Ablaufs werden nun zusätzlich die durch Forward-Checking veränderten Wertebereiche dargestellt.

```
C selected
  C = 0 consistent
    FC: domain(A) = {1,...,9}
    B selected
      B = 1 consistent
        FC: domain(A) = {2,...,9}, domain(U) = {1}
        A selected
          A = 2 consistent (Beachte: die Zuweisung A = 0,1 kommt hier
            FC: domain(U)={}, backtrack
          A = 3 consistent
          ... (identischer Verlauf wie in A=2)
          A = 9 consistent
            FC: nothing to change
            U selected
              U = 1 consistent  (U=0 kommt nicht mehr vor)
              result <- C=0,B=1,A=9,U=1
```

In diesem Beispiel ist der Suchraum der noch nicht belegten Variablen nach der Festlegung $C = 0$ und $B = 1$:

- $A \in \{2, \ldots, 9\}$, da $A \neq B$ und $A \neq C$
- $U \in \{1\}$, da $U = B$

Es müssen also die Varianten mit $A = 0$ und $A = 1$ sowie die Varianten mit $U = 9$ für alle Werte von A nicht mehr getestet werden, die Backtracking-Suche mit Forward Checking konvergiert also schneller.

Anmerkung: Die Festlegung von $B = 1$ und $C = 0$ hat auch Konsequenzen auf das erste Constraint, das man nun als $A+1 = 10 \cdot U$ schreiben könnte. Einfaches Forward-Checking überprüft nur Arc Consistency mit der zuletzt zugewiesenen Variable (also nur Costraints, die auch B involvieren). Erweitert man Forward Checking jedoch auf den AC-3 Algorithmus (→ Vorlesung) dann würde auch für jede Variable, deren Wertebereich verändert wurde (also nach der Zuweisung $B = 1$ wären das A und U) Arc Consistency überprüft. AC-3 könnte also sofort nach den Zuweisungen $C = 0$ und $B = 1$ erkennen, daß $U = 1$ und somit auch $A = 9$ sein muß. Es würde also praktisch ohne Suche konvergieren.